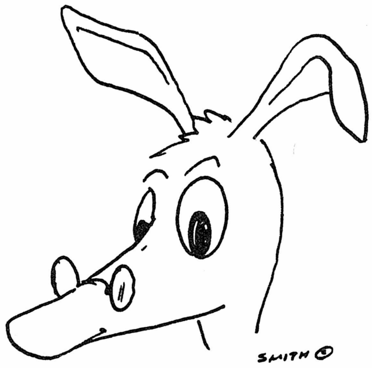
the AARDVARK JOURNAL

VOL. 3 NO. 2 JUNE 1982



*** INDEX ***

ARTICLE PAGE #
OS65D V3.3 by M.Heidt 1-2
COMMENTS ON V3.3 by R. Biendbach 2-3
WP-6502 RELOCATING by J. Roecker 3-4
MACH. LANGUAGE MONITOR 4-5
WPII 6502 by E. Rivera 5
SCREEN DISPLAYS by D. Tajkowski 5
HINTS ON "VIDEO SWAP" 5-6
FAST SCREEN CLEAR by R. Soderbeck 6-7
BAUD RATE SELECTOR by W.Zaydak 7
MUSIC FOR C1P by G. Artman 8-9
C1E INSTALLATION by E. Coohoon 10
RECOVERING PROGRAMS 10
DISK SWITCH HINTS by J. Scherr 10
C1S ROM TIPS by T. Warfel 11
INTERFACING THE 502 BD by J.Coyle 11
RE-LINE DELETE by A.Jansen 11
SAVING SPACE USING VARIABLES 11-12
CLUTTER FOR OSI by K. Lourash 12-13
TEXT EDITOR FOR C1E 14-15
CORRECTION TO ALIEN RAIN 15
"BUNNY" PROGRAM 15
CORRECTION FOR "YACHT RACE" 16
"EXPLOSION" PROGRAM 16
NOTE FROM C.DAVIES N.ZEALAND 16
CHANGES IN "BREAKTHRU" 16-17
CORRECTION TO ANTI AIRCRAFT 17
CLASSIFIED ADS 18

OS65D V3.3, A QUICK LOOK by M. Heidt

For almost a year now we have been hearing rumors of an improved 65D disk operating system (DOS) from OSI. Well, it has finally arrived. It is officially called OS65D V3.3. Many of the shortcomings of version 3.2 have been fixed, at a cost of 2K bytes of memory.

It is a bit misleading to say that only the DOS has been improved. Most of the changes are in the OSI version of Microsoft disk BASIC. Although the DOS is a totally separate entity from BASIC, they are so well integrated that many people think of them as one operating system. However, if you run some other language or program, such as Forth or the assembler, you still use the same DOS.

The familiar 65D kernel commands remain the same. The main change in the DOS is that the polled keyboard now emulates a Hazeltine 1420 terminal. This means that the strange shift key operation of V3.2 is gone. The DOS no longer calls the ROM at FD00. The keyboard now works more like a typewriter. Control functions are those of the Hazeltine. This should make programs from the big machines (e.g. WP3) work on polled keyboard machines. The rubout key replaces the shift-O as the way to erase a character from the input line. The auto repeat feature is gone. You must now press the repeat key along with the key you wish to repeat. This can be awkward for control keys when three keys must be held down. This can occur relatively frequently because the control keys are used to move the cursor when editing. The most visible change is that the cursor is now a flashing square which appears at the upper left corner or "home" position. The cursor moves down the screen and scrolling begins when the cursor reaches the bottom.

The most obvious enhancement to BASIC is the editor. This allows you to call up any line and completely edit it by moving the cursor to the appropriate place. There are shorthand commands to edit a line, recall the same line and to call the next line. This editor is always resident. There is also an editor program on one of the disks which will overlay the editor onto V3.2 BASIC.

The next big change is in the OSI BEXEC* program. This now includes most of the utilities that used to be individual programs under V3.2. The utilities now work the way they should. For example, they create function finds space in the directory for you. You still tell it how many tracks to reserve, but it will find the space on the disk for you. All of the utilities are presented as a menu when you boot the system. You can get a disk directory, create files, create data disks, (an initialized disk with an empty directory), copy disks, run programs, etc.

The copy program is worth special mention. This is the best copier I have ever seen for OSI disk systems. It allows copying with single or dual disks. It checks to see how much memory you have, and then reads as much as it can from the source disk before writing to the destination. In a 48K system with one drive it only takes three changes of the disk for a complete copy. The copier also initializes the tracks to be written. The only drawback I have found is that the copier always starts with track zero. It will stop anywhere you tell it, but you must start with zero. The old copier is still available if you wish to copy a few tracks in the middle of a disk. All of the utilities, including the copier, return you to the BEXEC* menu when finished.

The new diskettes also include some usefull utilities that are not in the BEXEC*. These are ones that you don't use quite so often. These include a very nice machine language re-numberer for basic programs, a repacker which removes extraneous spaces and remarks from basic programs, and a utility to add/delete or check for disk buffers on existing programs. All of the old V3.2 utilities are still there as well as one or two new ones. The new utilities are actually usefull where the old ones were more of an advertising feature.

When you get through playing with the utilities and sit down to write a program, you will find a wealth of new features in the V3.3 BASIC. You now have complete control of the cursor by use of special print statements. This gives the equivalent of the commands available in the Hazeltine 1420, such as HOME, CLEAR LINE, CLEAR TO END OF SCREEN, etc. You can define a print window anywhere on the screen. This window can be defined as any number of lines of any length, as long as it is smaller than the default screen size. It remains in effect until changed by another screen define operation or by a keyboard command from the command mode.

The new PRINT commands also permit control of color, screen size, and reading of the character under the cursor. You will also find the long awaited PRINT USING command. This lets you format numeric output so that decimals line up and columns are easy to define. No more converting numbers to strings just to line up a decimal. All in all, a powerful set of commands.

One of the most impressive things about the new DOS is the documentation. This is by far the best ever put out by OSI and as good as many in the industry. Although it is not as "slick" as that of the Apple or Radio Shack, I believe it is considerably more complete. For the first time DOS user, the manual starts at square one and guides you through the system step by step. The instructions are clear and there are adequate examples. The documentation does assume that the reader is sitting at the computer. The instructions take the FORM of "DO THIS", "YOU SHOULD SEE THIS", "IT MEANS THIS...".

For the more sophisticated user, the introduction refers you immediately to the section on V3.3. This section assumes that the reader is familiar with V3.2 and proceeds to explain the changes and enhancements of V3.3. At the back of the large binder is a set of appendices which summarize everything. This section can be pulled out and kept by the computer as a handy reference. Also included in the documentation are a complete memory map showing all versions of OS65D (old, new, 8-inch, and 5-inch) plus a disk map showing which is on each track of the disk.

While most of the information in the V3.3 documentation has been available in one form or another, this is the first time I have seen it all in one place and organized in a reasonable manner. It is also the best "first time users guide" that I have ever seen for the OSI system.

This should be enough to give you an overview of OS65D V3.3. It costs about $80 from most mail order places. I felt that the editor, copier and documentation were worth the price. The rest is gravy. I have not found any major bugs. Everything works if you take the time to read the manual and follow the examples. I am pleased with it and can only hope that OSI will see fit to make this software standard with all new systems. No OSI disk owner should be without it.

MORE COMMENTS ON V3.3 by R. Biedenbach

I'm not a subscriber to the Aardvark Journal, but a friend showed me the letters regarding OSI 65D V3.3. What I want to say is BALDERDASH!! AND MORE BALDERDASH! Just kidding guys, want to keep this light. But get out the fly paper and the raid for OSI 65D V3.3 is full of bugs.

I also believe it is the best software and documentation OSI has done to date. V3.3 has some very powerful features, such as print at, print using, windows, and the Find command which is my favorite.

When it comes to bugs even an Aardvark is not beyond reprove. I recently purchased Tiny Compiler on 8" disk and the program had missing lines and mis-numbered lines which resulted in the POKE and the LOOP functions from not operating. Even I, who recently wrote and published "OSI 65D V3.3 GUIDE" suffer from this human condition.

Here is a list of the bugs in OSI 65D V3.3:

1. CONTROL X DOES CRASH THE SYTEM
2. PRINTER DUMP DOESN'T WORK
3. ERRORS IN V3.3 BEXEC*
4. DISK COPIER WILL CRASH SYSTEM AFTER USE
5. V3.2 NOT UPWARD COMPATABLE TO 3.3 DATA FILES
6. V3.3 HAS LIMITED DOWNWARD COMPATIBILITY TO 3.2
7. OSI LEFT OUT ALOT OF VALUABLE DATA IN IT'S MANUALS.

Now comes the plug, my manual covers the above and more. So for all the answers, buy, read and tell your friends about "OSI 65D V3.3 GUIDE".

I will share this, to fix the control X so that it doesn't crash your system, (if you have less than 48K) add the following pokes to BEXEC*: POKE9593,234:POKE9594,234. This disables the keyboard routine from seeing a control X.

To enable the control X, POKE9593,201:POKE9594,24.

P.S. OSI 65D V3.3 is available from Buffalo Informational Tech. for $14.95.
209 Richmond Ave., Buffalo, NY 14222

RELOCATING WP-6502 by J. Roecker

WP-6502 is a very nice word processor for OSI machines. Unfortunatley WP-6502 uses memory locations $0222 through $0235 which are also used by many of the nonstandard monitor ROMs available today. I own a C1P which I have modified by adding an Aardvark C1S monitor ROM and an expanded screen. I recently wanted to write an article for the local OSI user's group and was asked to use WP-6502. The Aardvark C1S monitor ROM uses memory locations $0222 through $022F for cursor positioning and subroutines disabling WP-6502. The choices I had to enable me to use WP-6502 were: Put my standard OSI monitor ROM in, Relocate WP-6502, or Write the article by hand.

I have many routines which utilize the C1S monitor features so the first choice was eliminated. I did not want to write the article by hand so I chose to attempt to relocate WP-6502. A phone call to Aardvark T.S. reinforced my decision because Rodger Olsen laughed at my plans.

The items need to relocate WP-6502 are: (1) Some program to perform the relocation (OSI Extended Monitor), (2) A listing of WP-6502 (convenient but not necessary).

I used the following steps to relocate WP-6502 (all address locations mentioned are nonrelocated addresses):

1. Relocate WP-6502 to the desired locations. In my case I relocated it from locations $0222 through $0F98 to locations $0235 through $0FAB.

2. Replace the data which the nonstandard monitor ROM destroyed. In my case memory locations $0222 through $022F had to be replaced with the following data:

```
   $0222=52   $0223=02
   $0224=01   $0225=20
   $0226=80   $0227=40
   $0228=7F   $0229=7C
   $022A=5C   $022B=5D
   $022C=5E   $0220=0B
   $022E=23   $022F=5E
```

If you have an Aardvark C1E/C2E monitor ROM, locations $0230-$0234 will also have to be corrected.

```
   $0230=0A   $0231=42
   $0232=3C   $0233=0A
   $0234=00
```

3. Instructions which reference data outside of the limits of the relocation will not be modified by the relocation routine. In this case one such instruction exists; the instruction located at $024F. It will have to be modified by adding the amount of the relocation.

```
   $024F  BD0302  LDA  $0203,X
```

4. Data tables can cause relocation routines to misinterpret data for instructions. In the case of WP-6502 there are two cases of this; the instructions at locations $0671 and $0784. The address fields of these instructions will have to be modified by adding the amount of the relocation; $13 in this case. The non-modified instructions are:

```
   $0671  201503  JSR $0315
   $0784  207106  JSR $0671
```

5. BIT instructions are sometimes used to provide multiple entry points into subroutines. The relocation routine might possibly modify the address field of the BIT instruction, which is really another instruction. In the case of WP-6502 there are seven occurrences of this. Instructions located at $0307, $0317, $03D0, $03DD, $03EB should all be 2CA204 BIT $024A. The instruction located at $0323 should be 2CA202 BIT $02A2. The instruction located at $03C3 should be 2CA206 BIT $06A2.

6. The warm start code will have to be modified if you have a C1S because the C1S will mask out the 'Line Feed' character making it very difficult to edit data using 'Line Feed's. I modified the warm start code so it will use the old video drivers for all commands

except for the W/Tape command. It must use the new video routines. The following code has worked for me; insert it at the memory locations indicated.

```
0F3A A900    LDA #$00
0F3C 8D2906  STA #0629
0FCF 8D3906  STA $0639
0F42 A92D    LDA #$2D
0F44 8D1A02  STA $021A
0F47 A9BF    LDA #$BF
0F49 8D1B02  STA $021B
0F8F A929    LDA #$29
0F91 8D2906  STA $0629
0F94 A980    LDA #$80
0F96 8D3906  STA $0639
0F99 A969    LDA #$69
0F9B 8D1A02  STA $021A
0F9E A9FF    LDA #$FF
0FA0 8D1B02  STA $021B
0FA3 A920    LDA #$20
0FA5 8D2A02  STA $022A
```

7. The cold start code will have to be modified to use the proper data/text starting address. This address is $0F9B in the non-relocated WP-6502. This address will have to be modified by adding the amount of the relocation.

```
   0FAB A99F  LDA #$9F
   0FB8 A99B  LDA #$9B
```

The warm start jump will have to be modified also. The amount of the relocation will have to be added to the immediate data below:

```
   0FA7 A90B  LDA #$0B
```

This should complete the modification needed to allow WP-6502 to be relocated. I hope with these corrections you will be able to enjoy WP-6502.

The Hand Assembly of Programs for OSI's Machine Language Monitor

In this article I will discuss the implementation of assembly language programs on the C1P by the use of the machine language monitor. All numbers will be in hexadecimal, and a knowledge of the monitor and its capabilities is assumed. If you aren't familiar with it, I suggest you read page D-5 of the Users manual.

Although BASIC is great, machine language offers a tremendous challenge, and it is far faster and slightly more flexible than BASIC. A knowledge of machine language is useful to everyone since it can be used to increase the speed and power of BASIC. The monitor provides a rudimentary, though adequate, tool with which to explore and learn about machine language.

Since almost any 6502 book deals exclusively with assembly language, and, ultimately, any machine language program starts out in assembly language, it is the user's problem to convert from assembly mnemonics to hexadecimal opcodes that can be used by the monitor. If you will look at a conversion chart, such as the one on the back of the Users manual, you'll find that under any given mnemonic, several opcodes are yours to choose from. This is very confusing and it merits some explaining.

Each mnemonic converts directly into one opcode, but most require one or more additional bytes which determine the informtion that the microprocessor is going to process in the manner that the opcode dictates. For instance "JMP A274" tells the microprocessor to start executing instructions at location A274. The opcodes that you would put in with the monitor to get it to do this are (the critter doesn't understand JMP unless you have an assembler program) When the microprocessor finds a 4C, it automatically knows that it needs to get the next two numbers, which will be the address.

Similarly, when it executes the instruction "LDA, FF" which translates to "A9 FF" it knows that the instruction "A9" needs only one number to operate on and the next number after FF will be the next instruction On the other hand, the instruction "LDA (DF00)", which translates to "AD 00 DF" tells the microprocessor to load into the accumulator the number that is stored in location DF00. In the first instance, we loaded a literal number into the accumulator, the one immediately after the instruction. This mode of addressing requires two bytes, one for instruction, one for "argument". Consulting the last page of the Users manual, you will find that the opcode at the intersection of the "immediate" column and the "LDA" row is "A9". In the second example, since we wanted to load in the value of a specific memory location, we used the opcode "AD" to tell the microprocessor that we were going to use the "absolute" form of addressing. Since there are thirteen forms of addressing, you should read the chapter in your 6502 book, that explains about addressing. This is pretty deep stuff, so you might want to read it over several times. In addition you should become familiar with the way each form is represented in an assembly language program.

Now, assuming that you understood all that, try your hand at converting and loading this simple assembly language program:

```
STA D210  ;store the value of  the
accumulator in the center of the scr
JSR FD00  ;get  a  character  from the
keyboard which  will  be left  in  the
accumulator
CMP   4D       ;see if the character is an
"M"
BNE F9     ;start over if it isn't
JMP FE00  ;return program control to the
monitor
```

If you translated that program properly it will look like this:

```
8D 10 D2
20 00 FD
C9 4D
D0 F6
4C 00 FE
```

All of which brings us to another point. When you come to the end of a machine language program, you still have to have the machine doing something. If the program is a USR routine, you must end it with the opcode RTS (60) to get back into BASIC. If the program is independent of BASIC, like this one or any of the examples that you will enter from a book, you probably want to return to the monitor as I did here. Unlike BASIC, your machine language programs can be put anywhere in RAM, even in the screen memory, except 00FB-FF, which are used by the monitor.

If you don't have a tutorial book, get one! I also highly recommend Aardvark's FIRST BOOK OF OSI, which tells where all of BASIC's useful subroutines are, and how to use them (it also has some great stuff about how to use BASIC). A disassembler program, which, as the name suggests, converts opcode to mnemonics would also be very helpful since it would enable you to learn by example by looking over other peoples' programs and the BASIC machine code.

EFRAIM RIVERA, AURORA, COLORADO

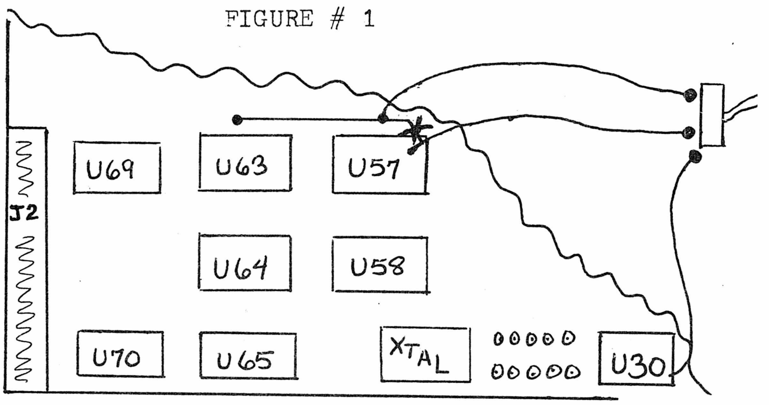
I purchased WP-6502 a word processor program from DWO QUONG FOK LOK SOW and was totally satisfied with it, except for one minor problem. Data was transferred to the printer at 300 baud. I would print 3 pages of information and it would take quite a long time. After some investigation I found that the word processor program used the ROM routine to initialize the ACIA. Even though it is possible to use basic to poke the necessary values to the ACIA to set it up for 4800 baud, as soon as I ran my word processor program it would re-initialize for 300 baud. After looking at the C1P/Superboard schematics I found an easy way of modifying the buad rate in hardware. The modification includes installing a switch so that the operator can switch between 300 and 4800 baud, the 300 baud is still needed for the cassette.

To install the modification follow these steps: (SEE FIGURE #1)

1. Cut the trace going to U57 pin 2.
2. Solder a wire to U57 pin 2, solder the other end of the wire to the common terminal of the SPDT switch.
3. Solder a wire from the trace that was cut to one pole on the switch.
4. Solder a wire from U30 pin 14 to the other pole of the switch.
5. Mount switch on front panel at any convenient location.

NOTE*: the above modifiction should be installed on the COMPONENT SIDE of board.

FIGURE # 1



DAVID A. TAJKOWSKI

For those owners of Series II C1P's that like watching mindless displays on their screens, might I suggest one of these two, the second of which makes my eyes hurt.

```
A) 10 FORA=0TO255:FORB=53314TO56999:
POKEB,A:NEXTB,A:GOTO10
B) 10 FORB=53314TO56999:A=INT
(RND(B)*255):POKEB,A:NEXT:GOTO10
```

I am not certain if they work the same way on the old C1P.

HINTS ON "VIDEO SWAP"

I have a Superboard II series II and am very impressed with the improvements. The digital to analog converter is a real potential superport. The 12 X 48 display is a fantastic improvement in readability. I use it almost exclusively for programming now that I have converted some of my utilities to this format.

I have been watching the Journal and other publications for program or hints on these features but to no avail. So maybe I can get the ball rolling by sharing some of my thoughts on the 12 X 48 display.

The 12 X 48 display is enabled by a "Video Swap" program which OSI supplies with the C1P. The Series II Superboard is identical so I got a copy from my dealer to use with the Superboard. It comes as a BASIC program that loads in the highest available memory a replacement for the video driver routine in ROM. It's hooked to BASIC via the output vector at 538-539.

I soon discovered some difficulty with this program. First of all it is very volatile. Using the Break Key returns to the normal 24 X 24 format and seemingly looses the program. I soon discovered it writes over the output vector and resetting this recovers it. In an 8K machine you use POKE538,73:POKE539,31. These must be one one line because you can send your

CPU on a wild goose chase if you have only set one of the two values. If you have other than 8K you can PEEK these locations while in the 12 X 48 mode to find the proper values.

My next dilemma concerned using this with other machine code routines. OSI has made this a relocatable routine but the BASIC program sets it by the highest memory in your machine. Quite a few of my machine code routines were set here also and I had to make a choice about which to use. I first used a little trick to get these programs both in. I first checked with the machine routines to see where they started. Next I used the memory limit (BREAK C) to set high memory before loading the "VIDEO SWAP" then loaded my second program. Kind of a long process, but it worked.

LO the next problem. It seems as if a number of my routines also used the same output vector. This is normally set to $FF69 and I found that these programs were jumping to that address. Since I had replaced the output vector for the video swap I had to set these to the Video Swap also. An indirect jump via the output vector would be the best way.

I still had the difficulty of loading the swap after I had a program in. It had to be in first. This I solved by re-numbering the video swap with higher line numbers and removing the new command. (In line 900 in the OSI version.)

Now I could load it with a BASIC program already in memory. However, it used up quite a bit of free RAM this way. Then I converted it to machine code load with the extended monitor (see Journal Vol. 1 No. 5, page 8). I also added the following bit of code so it comes up in the 12 X 48 format. With this addition I can go to $0222 and reset the pointers after a BREAK.

```
10 ;SET POINTERS FOR VIDEO SWAP
20 ;VALUES FOR TOP OF 8K ($1F47)
30 *= $0222
40 0222 A9 47     LDA #73    ;LOW BYTE
50 0224 85 81     STA 129    ;STRING TOP
60 0226 85 85     STA 133    ;RAM LIMIT
70 0228 8D 1A 02 STA 538    ;OUTPUT
80 022B A9 1F     LDA #31    ;HIGH BYTE
90 022D 85 82     STA 130    ;STRING
100 022F 85 86     STA 134    ;RAM
110 0231 8D 1B 02 STA 539    ;VECTOR
120 0234 A9 01     LDA #1     ;SET TO
130  0236 85  FB        STA 251     ;12X48
SCREEN
140 0238 4C 47 1F JMP $1F47 ;VIDEO SWAP
```

NOW..ANYONE WITH SOME PROGRAMS FOR THE D TO A CONVERTER? LET'S HEAR IT

R. SODERBECK, JACKSON MICHIGAN

Some time ago I found a VERY FAST SCREEN CLEAR for the C1P, as follows:

```
10 A=PEEK(129):B=PEEK(130):
POKE129,0:POKE130,212:S$=" ": FORS=1TO7
20  S$=S$+S$+" ":NEXT:POKE129,A:
POKE130,B
```

I tried it on my C8P and it only cleared part of the screen. I experimented with the program and came up with the following to clear the entire screen on the C8P.

```
5   REM:VERY  FAST SCREEN CLEAR FOR THE
C8P
6  REM:BY R. SODERBECK 3-25-82
10  A=PEEK(129):B=PEEK(130):X=0:
Y=216:Z=21:S$=" ":T$=" "
20  POKE129,X:POKE130,Y:FORK=1TOZ:
S$=S$+T$+T$+T$:NEXT
30 POKE129,A:POKE130,B
```

There are other combinations, but this one works very good.

I was interested in finding out how this program worked and after some experimenting, I came up with some interesting results.

The screen memory, for the C8P, is at D000 through D7FF and the color memory is at E000 through E7FF. I found that the required POKE location in the screen clear program; POKE129,0:POKE130,216 (HEX D800); is one address beyond the last screen memory address of D7FF. The equation of line 20 builds a series of strings using the values of S$ and T$ (in this case, blank spaces) starting at the "POKED address minus 1, and working backwards.

To see how this works, use S$="X" and T$="O" (for example) and very the value of Z from 1 up. You may even change the number of T$'s in the equation, however, if any one loop through the for-next loop creates a string longer than 256 characters, you get a "long string" error. By changing the POKE address, the equation, and the values of S$ and T$, you can put most anything, anywhere on the screen.

This also works for the color memory. However, I found that it works only for color values of 0-9. After trying various ideas, I found that the color values can be represented by letters; 1=A, 2=B, 3=C,etc.

The following creates color patterns demonstrating some of the possibilities using this program. The variables used are: C$=color; X and Y=starting location; Z=number of FOR-NEXT loops; L=number of lines; N=address increment for start of next line; B=equation; C=line length change; G=number of times through delay loop between lines.

```
10 REM:COLOR DEMO BY  R.  SODERBECK
3-25-82
100 POKE56832,5
105  X=0:Y=216:Z=21:S$=" ":T$=  "
":GOSUB5000
110 Y=232:S$="B":T$="B":GOSUB5000
140 C$="A":X=212:Y=227:Z=8:L=12:N=-63:
B=1:C=0:G=500
142 GOSUB500
145  C$="J":X=32:Y=225:Z=1:L=16:N=64:
B=2:C=1:G=50
147 GOSUB500
150  C$="N":X=229:Y=229:Z=1:L=12:N=-63:
B=2:C=1:G=300
152 GOSUB500
155 C$="D":X=126:Y=226:Z=12:L=6:N=-64:
B=1:C=0:G=700
157 GOSUB500
160  C$="H":X=182:Y=226:Z=1:L=20:N=64:
B=2:C=1:G=10
162 GOSUB500
165  C$="F":X=94:Y=231:Z=7:L=8:N=-64:
B=1:C=0:G=200
167 GOSUB500
170 C$="L":X=142:Y=128:Z=6:L=12:N=64:
B=1:C=0:G=100
172 GOSUB500
190 GOTO105
500 FORA=1TOL
505 S$=C$:T$=C$:R$=C$
510 ONBGOSUB1000,2000
520 X1=X+N:Z=Z+C
530 IFX1<0THENGOTO570
540 IFX1>255THENGOTO560
550 X=X1:GOTO580
560 X=X1-256:Y=Y+1:GOTO580
570 X=X1+256:Y=Y-1:GOTO580
580 FORF=1TOG:NEXTF,A
590 S$=" ":R$=" ":T$=" "
599 RETURN
1000 POKE129,X:POKE130,Y:FORK=1TOZ:S$=
R$+T$:NEXT:RETURN
2000 POKE129,X:POKE130,Y:FORK=1TOZ:S$=
S$+D$:NEXT:RETURN
5000  POKE129,X:POKE130,Y:FORK=1TOZ:S$=
S$+T$+T$+T$:NEXT
5010 RETURN
```

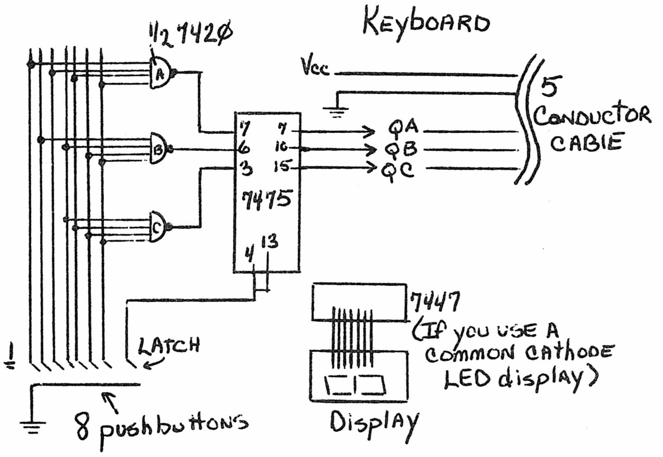
WILLIAM ZAYDAK, ROCHESTER NEW YORK

Similar to Dave Broyhill's idea about a baud rate selector, you will see a schematic for a circuit addition that I made about a year ago on my SBII Series II. It has been in use with the baud mod on U63 that most people know about. The addition came out of the sheer frustration that came from debugging 6K programs. The idea was that in the SAVE mode if I tapped off the different clock rates available then I could scan through the program LISTing or RUNs at different speeds. It also allows me to stop LISTing and RUNs without having to re-enter line numbers, or commands to continue.

The output of the 74151 is selcted by a bit pattern that is present at the '151's select (ABC) inputs. To hold the bit pattern present a 7475 latch was used. It will latch when the latch enable goes from H to L to H. The input to the '75 can come from an encoder like the one on the schematic or a set of switches. The encoder only has to go up to 7; notice the '151' pin 4 has no connection on it. This is so that when the 0 (latch) key is pressed, the multiplexer will switch to a null input; no clock pulses present-the display will stop. The latch (0) key setup was opted for over detecting a key closure (a 7430 connected onto the key lines with the other 1/2 or the 7420 on its output as an invert and creating a latch pulse with a 74123. This way accidently pressing a key will not change the baud rate. You must press the baud rate key you want then press and let go of the latch (0) key. As for the clock lines, just trace from the chip pins listed to a through hole. In my final version I must admit to soldering onto the U59 and U60 pins from below with a 27W iron and tinned wire. I mounted the '151' on perfboard, on the 600 board. The baud rate keyboard and encoder connect to the '151' with a 5 conductor cable. You could use a DIN plug and make up a header for when it is disconnected like I did.

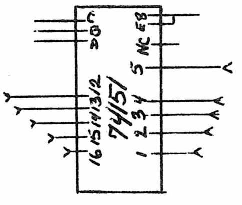
There is a C1P-MF in the lab where I teach and I recently had the experience of finding a data track eaten away. I made the mod suggested by Ed Keating, it is so simple and it works; why doesn't OSI set it up that way? The option should be to modify it so that the head is always loaded, not the other way around.

A statement that I have found useful withthe C1P-MF is: POKE9800,254;?:?:?:POKE9800,64. It came from taking Dave Sugar's suggestion to play round with those pokes he listed. This statement will give you rather effective screen clear and will space simultaneous ouput to a printer with 3 linefeeds.



NO FOIL CUTS!

ON 600 BOARD:



INPUTS:

0	NC
1	U59 Pin12
2	U59 Pin13
3	U59 Pin14
4	U60 Pin14
5	U60 Pin13
6	U60 Pin12
7	U60 Pin11

Output:

Q goes to J3; the Molex where the power wires go into the board. Disconnect whatever is on P8 Connect in Q from the 151.

MUSIC FOR THE C1P ET. AL.

by Gerald Artman

After trying to make more than the sliding whistle sound listed in the C1P manual (Series II), I have written the following program to write and play music through the noise port. For those with Series C1P's, you can buy instructions on how to add the noise port from Aardvark. Those with other micros can refer to the May 81 edition of "MICRO" for instructions on adding a noise port. Basically, any mic with a parallel port can use the routines.

The program produces a square wave output. For those who are confused already, square means that the sound starts and stops abruptly rather than rise and/or fall slowly. To produce this sound you only have to toggle a line on and off. A noise port produces a very high pitch sound. This pitch can not be altered. On the C1P it can be played loud or soft, depending on what value is sent to the port. To make a range of notes, the computer must turn off and on the sound at different lengths of time. The result-a lower note. Because BASIC operates through an interpreter it is usually too slow to change the output significantly.

To make simple music you must use machine language routines to divide up the time and play the notes. Most 6502's operate at a clock rate of 1000000 cycles per second. To produce a tone, the frequency will be the result of the time spent in a machine language routine divided into the clock rate; FQ = 10(6)/LOOP TIME. The length of time the note will be played will be dependent on the number of times the loop is executed; N X LOOP/10(6).

Table #1 is a list of the frequencies from middle C to B. To produc other octaves of higher or lower notes just divide or multiply by two for each octave. The following equation is used to produce a note arguement FQ in the range 0-255:

$$FQ = \frac{(10^6/\text{DESIRED FREQ.}) - 45}{70}$$

The 45 accounts for the time used in the operation of the loop. The loop is expanded from the smallest possible, 5 cycles, to 70, in order to produce low notes. Figure #1 shows the notes that can be produced by this equation.

To alter the duration of play, an additional parameter must be computed. Music is played according to tempo with each note having a beat value. Common tempos are 100 beats/minute for 4/4 time and 80 for 3/4 time. The actual time duration for a note would be:

INSERT $$DU = \frac{(\text{BEAT VALUE}) \text{ X } 60}{\text{TEMPO}}$$

The number of times to execute a loop can be reduced to the equation:

TIME ARGUEMENT = DU X FREQUENCY

This music program calculates these arguements for each note and stores them in memory for play or saving on tape. Rather that use array storage or strings, each value is poked into memory directly starting in the upper 4K (1000 hex). The machine language routines are located at $0230 to $02B2. If this space is not available on your machine, they can be relocated. Do not forget to change the calling address in the basic program. To save a song on tape the program puts a header on the tape then prints the peek value of each memory location until the end of the table. On input the program reads each value and pokes it directly into memory. Provisions are made for recording sections and adding them together.

When using the program, a menu will appear. To input a series of notes, first the tempo will be requested. Then the beat value, octave and note name for each note will be requested. Each octvae is numbered from 1-5, see Figure #1. For rests use octave 3 note name O. To exit the input section enter END for note name. To correct a mistake count to the note number to be changed. The edit function will recalculate and replace the information. As written the program will handle about 1000 notes. I haven't typed in that large a song, but I have entered five songs in a row. By relocating the storage area, music can be added to other basic programs provided there is enough room. Each note uses three bytes. All that is needed to play the music is the machine language routines and the note table. Look out Atari.

LISTING 1

NOTE ARGUEMENT PASSING PROGRAM

```
START AT$027A
USES ZERO PAGE LOCATIONS: $FO,$F1,$F2,$F3,$F4

027A LDA #00                  ; set zero page for indirect at $1000
027C STA $FO
027E LDA #$10
0280 STA $F1
0282 LDA #$FF    (loop)       ; store loudness in $F2
0284 STA $F2
0286 LDY #00                  ; set y to zero for indirect address
0288 LDA ($FO),Y              ; get duration low arguement
028A STA $F3                  ; store at $F3
028C INY                      ; increment for duaation high
028E LDA ($FO),Y              ; get duration high and store
0290 STA $F4
0292 INY                      ; increment for note arguement
0293 LDA ($FO),Y              ; get note
0295 BNE 02                   ; if not 0 then skip next step
```

```
0297 STA $F2                   ; else turn off loudness for rest
0299 CMP #01                   ; see if end of song
029B BEQ (out)                 ; if so return to program(basic)
029D JSR $0230                 ; else play the note
02A0 INY                       ; bump y and add to table base
02A1 TYA
02A2 CLC
02A3 ADC $F0
02A5 STA $F0
02A7 BCC 02
02A9 INC $F2
02AB LDX #$FF                  ; waste some time to seperate notes
02AD DEX
02AE BNE FD
02B0 JMP (loop)                ; get next note
02B3 RTS        (out)          ; return to basic program
```

NOTE: CAN BE RELOCATED ANYWHERE. MUST CHANGE ADDRESS AT 029D TO NEW SUBROUTINE LOCATION AND BASIC PROGRAM CALLING ROUTINE (X=USR(X)) BY POKEING NEW LOCATION IN 11 and 12.

LISTING 1 con't.

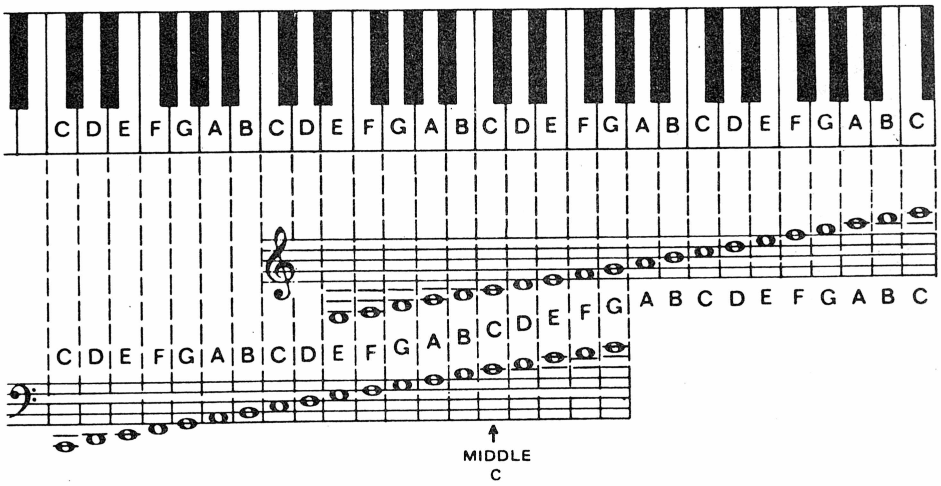
PLAY SUBROUTINE START AT $0230

```
0230  LDX $F2   (start)        ; get loudness. 00=rest  FF= note
0232  STX $DF00                ; send to port
0235  TAX                      ; move frequency arguement to counter
0236  DEX         (loop)       ; count down to zero
0237  NOP NOP NOP NOP NOP      ; add 30 states to loop. total time
023C  NOP NOP NOP NOP NOP      ; is 35 cycles
0241  NOP NOP NOP NOP NOP
0246  BNE (loop)               ; if not 0 do over
0248  BEQ 00                   ; waste time to equal duration update
024A  BEQ 00
024C  BEQ 00
024E  BEQ 00
0250  BEQ 00
0252  LDX #$00                 ; turn off port
0254  STX $DF00
0257  TAX                      ; move frequency arguement to counter
0258  DEX         (loop2)      ; count equal time with port off
0259  NOP NOP NOP NOP NOP
025E  NOP NOP NOP NOP NOP
0263  NOP NOP NOP NOP NOP
0268  BNE (loop2)              ; if not 0 do over
026A  DEC $F3                  ; count down duration low arguement
026C  BNE (wait)               ; if not 0 waste same amount of time for
026C                           ; high countdown
026E  DEC $F4                  ; count down duration high arguement
0270  BNE (start)              ; if not zero start over
0272  RTS                      ; else return
0273  BEQ 00     (wait)        ; waste time equal to duration high check
0275  BEQ 00
0277  BNE (start)              ; low not zero, keep going
```

TABLE 1

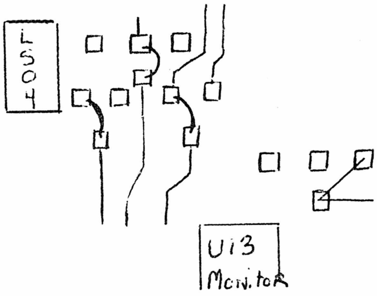
Middle C	261.62
C#,Dflat	277.18
D	293.66
D#,Eflat	311.13
E	329.63
F	349.23
F#,Gflat	369.99
G	391.99
G#,Aflat	415.30
A	440
A#,Bflat	466.16
B	493.88

Figure 1


C D E F G A B C D E F G A B C D E F G A B C D E F G A B C
C D E F G A B C D E F G A B C D E F G
A B C D E F G A B C
MIDDLE C

INSTALLATION OF C1E by E. Cohoon

There must be at least three (3) versions of the monitor rom from OSI. In my Superboard I had an EPROM labeled B2758 which is an Intel 5 volt 8K device. I chased around in circles for a short time until I verified that fact. After that it was a very short hop to changing W8 from ground to A10. No splices, cuts, etc. needed. Just change as diagramed below:



A hint when working with the pads. I placed the point of my solder iron under the jumper loop. After the solder melted I just lifted up. Came out with no problems. I then cleaned the holes with solder wick. I took the pins out of an Augat IC socket I had and soldered them into the pads. No. 22 wire worked just find in them for readily changeable jumpers. The pins look as follows 4X: A little scrounging around should turn up suitable substitutes.

The cursor functions are great in the C1E. Almost as good as the HP-85. Unforutnately, that is all I have been able to use so far.

NOTE** In reference to the PARALLEL PRINTER Interface article in the Dec. 1981 Journal, we didn't realize that Australian dollars are worth a bit more than yours, so we are now updating you with current prices in Aus. and U.S. currency, also it is now avail. assembled and tested, either with a wire wrap plug (straight in the ACIA socket) or using a ribbon cable.

PARALLEL PRINTER INTERFACE PRICES
UNDRILLED PCB.......$4-50 $5.95 U.S.
DRILLED PCB.........$5-60 $7.16
ASSBLD AND TESTED...$39-95 $44.95
(WIRE WRAP PLUG)
ASSBLD AND TESTED...$37-95 $42.50
24 WAY CABLE W/DIL
PLUGS (8 INCH)......$11-95 $12.95
ADD $2 TO ALL PRICES FOR POST. AND PACKING.
WRITE: GEOFF COHEN, 72 SPOFFORTH ST., HOLT, A.C.T. 2615, AUSTRALIA

JOHN C. SCHERR, VIRGINIA

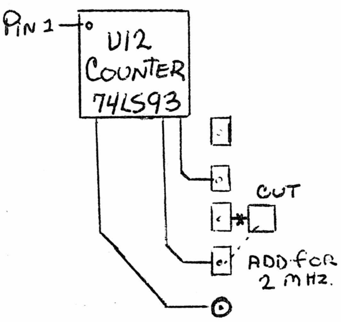
I have just installed the Disk Switch which I purchased from Aardvark. The switch works as advertised by turning off the drive motor when not needed. The kit took me about 2 hours to wire and I had no trouble with the assembly.

I did have trouble when using your Fantastic Copy. The motor would never come on. To fix this I added a DPST switch which reconnect the foil cuts at E and I.

RON BATTLE, BOZEMAN, MT

In the October 1981 Aardvark Journal, Thomas Owen of Miami Florida offers a fix for disk when a C1P has been modified to operate at 2 Mhz. His circuit will work but is not needed because a 125 Khz signal is already available on the 610 board. Pin 11 (eleven) of chip U12 outputs 125 Khz when the input clock is 2 Mhz (O(2)). OSI even provides a jumper pad for this fix. Just cut the original jumper and install a new jumper wire as per diagram #1. Diagram #2 illustrates the actual frequencies coming from chip U12 when the C1P is running at 2 Mhz. I urge C1P owners to purchase the Sams service manual, my source of information.

DIAGRAM 1 (610 BD)



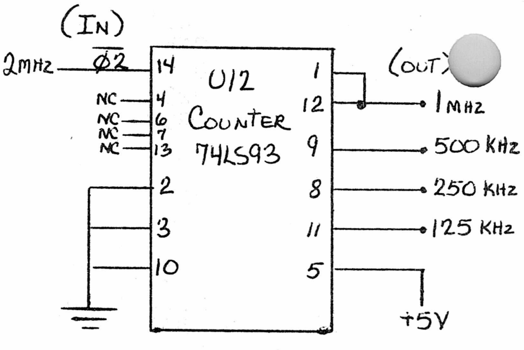


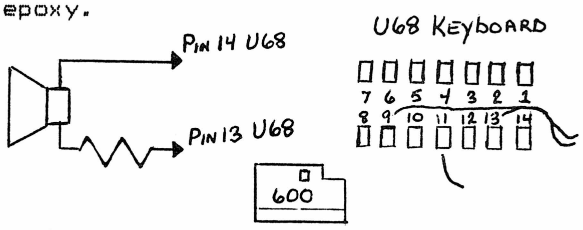
DIAGRAM 2 (610 Bd)

A. MOSSBERG, FLORIDA

If you've messed up a program, particularly one that POKEs alot to low memory, you can recover the program by hitting BREAK (M), .A274, and list the program. Or if there is garbage after a few lines, notice what line number was last listed, re-enter BASIC again by jumping to A274 through the monitor. Type the line number to erase it, if the screen goes crazy go back to BASIC through A274. Type list, if the program appears correct save to tape and cold start. If the program still has garbage repeat steps to remove it. Mix yourself a drink and be glad you didn't lose all that work.

A FEW TIPS FOR C1S ROM OWNERS
by Tom Warfel, Ohio

If you have the new Aardvark ROMs and want to add the bell, but don't have a 7417, take heart, you don't need it. Connect the speaker and resistor to where pins 13 and 14 of U68 would be. I would suggest adding a switch as somtimes you may not want the bell. I mounted the speaker behind the keyboard with epoxy.



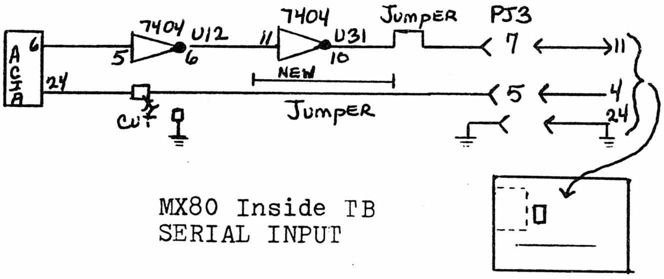
If you have the Semi-intelligent Terminal program and would like to use the new screen drivers, add:

```
925 POKE632,238:POKE633,255
```

Also if you're considering adding the Video Mod II, fig. 3, U29 should be marked U44. (74LS163)

INTERFACING THE 502 BD (C4P) TO THE MX-80
by J. Coyle, Pennsylvania

Instead of populating your Challenger for RS232 and investing in the Epson RS232 Interface I suggest tieing the existing TTL Serial Input of the Epson to the 502 board.



MX80 Inside TB
SERIAL INPUT

Also jumper MX80 TB Pins 5, 15, and 21 to Pin 24 for 300 baud. Pin 21 is to enable Serial Input. Pins 3, 5, 17 and 15 vary baud rate. Pin 24 is ground.

BAUD	PIN			
	3	5	17	15
75	L	L	L	L
110	L	L	L	H
134.5	L	L	H	L
150	L	L	H	H
200	L	H	L	L
300	L	H	L	H
600	L	H	H	L
1200	H	L	L	L

RE-LINE DELETE FOR OS65D
by A. Jansen, S. Australia

We have discovered that by printing line numbers to memory and then using the indirect file to bring them back into the workspace an intelligent LINE DELETE results. For example:

```
POKE9554,120:POKE9368,120:REM  SET
INDIRECT FILE (80 FOR 24K)
POKE9105,0:POKE9106,120:REM SET MEM
OUTPUT TO SAME PLACE
FORL=2001TO3201STEP10:PRINT#5,L:
NEXT:?#5,"OK"
```

Print line #'s 2001 to 3201 in steps of 10 to mem. The "OK" is a dummy to cause a 'SN ERROR' otherwise garbage will be printed to the workspace. The (CTRL X) to read these #'s back to the workspace and they are deleted from your program! You must then (SHIFT M) as usual to reset the indirect file. You must also reset the memory pointers if you want to repeat. Using this, any block of lines you can specify can be deleted at will. We normally write programs with line #'s ending in 10, then make corrections on even lines, reserving odd lines for diagnostic prints etc. This delete then allows us to remove the prints and any other gargabe without affecting the program.

SAVING SPACE USING INTEGER VARIABLES
by P.E. McQueen, Indiana

Ever wonder why some variables in some of the OSI utility programs use a "%" sign at the end? Although the OSI documentation doesn't mention it (among many other exceptions) it is part of the OSI BASIC.

The "%" sign indicates that the variable is an integer variable and as such requires less space to store each separate occurrance such as in an array. While a standard numeric variable requires five (5) bytes to store each separately defined occurrance, the numeric integer variable can be stored in two bytes. I was going to provide a copy of a short program to illustrate this but half the fun of workin with micros is to try things out for yourself and not having to be led down the primrose path every time. A few hints for those who may just be discovering what variables are. The use of the DIM statement and the FRE(X) command will help in illustrating the problem.

The numberic variables (such as x,y,i,...etc.) use five bytes to store any value (including floating point) up to a maximum of nine significant figures. The values are actually stored in BCD and it would appear that 10 digits could be stored in the five available bytes but the other half byte is used to store the sign (+ or -) of the value. The storage of the numeric integers is in binary format and is limited to a maximum value of 32K (32768 bytes). Since there are 16 bits in the two byte storage area it would appear that values to 64K could be stored but again one of the bits is used for the sign so only 15 bits are available to store the value. Also, by definition, integer variables are not floating point and values in excess of 32K will cause a data error. One limitation. These variables will not work in a FOR NEXT loop. For three bytes gain, it probably does not seem to be worth the effort to use the integer variables but in large arrays the bytes add up in a hurry.

The article above concerns the use of integer variables to reduce program size where applicable. Considerable space can be saved if arrays can be constructed with integer data instead of decimal data. The following listing is a short program that displays a figure of a man that is very important in the historical development of our country. To achieve the desired effect, the display must be done in B & W. It also is the basic technique to display any figure that can be defined in varying shades of grey.

```
2  FORI=1TO32:PRINT:NEXT
5   POKE56832,4:REM TURN COLOR ON  32
CHARACTERS
7  L=64:DIMX(19),Z(19)
10 INPUT"INPUT BACKGROUND (99=END)";J
14 IFJ=99THENPOKE56832,1:END
15 ST=57344
30 FORI=STTOST+2048
40 POKEI,J
50 NEXTI
60 INPUT"SATISFIED";Y$:IFLEFT$(Y$,1)<>
"Y"THEN10
65 FORI=1TO32:PRINT:NEXT
70 D=ST+6+6*L
80 FORJ=1TO19:READX(J),Z(J):NEXTJ
90 FORJ=1TO19
100 D=D+64
110 GOSUB5000
120 NEXTJ
3500  I=PEEK(57100):IFI=PEEK(57100)
THEN3500
4000 RESTORE:GOTO10
5000 FORI=D+X(J)TOD+Z(J)
5010 READS:POKEI,S
5020 NEXTI
5030 RETURN
10050 DATA6,8,4,9,3,10,2,10,2,11,2,11,1,
11,1,11,1,11,2,10,2,10
10060 DATA1,10,1,10,1,10,1,10,1,10,1,12,
1,12,1,12
10100 DATA15,2,3
10200 DATA3,14,14,14,14,14
10300 DATA15,14,15,3,14,14,14,14
10400 DATA15,14,2,7,7,3,15,14,14
10500 DATA14,14,15,3,3,3,3,15,15,2
10600 DATA14,14,2,15,3,3,3,3,3,14
10700 DATA15,14,14,2,15,2,14,15,15,14,
14
10800 DATA15,3,2,2,3,15,14,3,2,14,15
10900 DATA3,15,3,2,3,3,3,7,3,15,3
11000 DATA15,3,2,15,3,3,15,15,2
11100 DATA3,2,2,2,15,15,3,14,2
11200 DATA3,3,15,2,14,2,2,15,14,2
11300 DATA3,3,15,2,14,2,2,15,14,2
11400 DATA3,3,2,3,14,14,14,14,14,15
11500 DATA3,2,2,2,7,15,2,15,15,15
11600 DATA14,14,14,14,14,14,14,2,15,14
11700 DATA14,14,14,14,14,14,14,14,14,14,
14,14
11800 DATA14,14,14,14,14,15,15,14,14,14,
14,14
11900 DATA14,14,14,14,14,14,7,7,7,14,14,
14
```

CLUTTER for OSI

After doing some immediate mode calculations and making a few mistakes, I was faced with an all-too-familiar problem: the screen was cluttered with "OK's", error messages, and blank lines. I had one key screen clear available, but there was valuable information on the screen that I didn't want erased.

The "CLUTTER" program solves this C1P-C4P dilemma. It erases blank line, OK's, and error messages, and transfers the remaining lines to the bottom of the screen. It can be set up as a USR routine with POKE11,34:POKE12,2. Call Clutter by typing X-USR(X), or put X-USR(X) in line zero and call the Clutter program with a RUN command. higher line numbers will not be executed because Clutter exits to the immediate mode.

Clutter has two pointers; start-ST, and poke point-PP. The start point is moved up the screen, line by line. The start of each line is examined to see if the line should be erased or not. If the line contains character strings identical to those stored in Clutter's data table (TBL), the line is erased. If the line is to be saved, it is stored in the location pointed to by PP. When Clutter is done, it prints an $A4 graphics character (white block) and jumps to the immediate mode.

By adding to the table at the end of the program, you can expand Clutter's repertoire of lines to be erased. There must be a null (0) after every entry and a double null at the end of the table. By changing BNE DECST (near the start of the program) to NOP NOP and LDA #$62 (in the DONE routine) to RTS, it's possible to selectively erase and/or pack screen lines in a running BASIC program.

For users with non-standard screen formats, LEN should be set to the number of characters per line that your screen displays. One last word: before trying out this program, make sure you have it copied correctly! I had many a systems crash when incorrectly designed versions of Clutter ran amok in RAM.

```
 10 0000            ;CLUTTER FOR OSI
 20 0000             LINE=$20           ;$40 IF C2/4
 30 0000              LEN=$18           ;$40 IF C2/4
 40 0000               ST=$45
 50 0000               PP=$47
 60 0222                *=$0222
 70 0222            ;
 80 0222 A965              LDA #$65      ;#$40 IF C2/4
 90 0224 8545              STA  ST       ;SET START, POKE POINT
100 0226 8547              STA  PP
110 0228 A9D3              LDA #$D3      ;#$D7 IF C2/4
120 022A 8546              STA  ST+1
130 022C 8548              STA  PP+1
140 022E D034              BNE  DECST    ;BRANCH ALWAYS
150 0230            ;
160 0230 A2FF       CKLIN  LDX #$FF
170 0232 A0FF       C0     LDY #$FF
180 0234 E8         C1     INX
190 0235 C8                INY
200 0236 BD8302            LDA  TBL,X
210 0239 F03D              BEQ  ERASE    ;IF NULL, ERASE LINE
220 023B D145              CMP  (ST),Y   ;COMPARE CHAR. TO SCREEN
230 023D F0F5              BEQ  C1       ;LOOP IF A MATCH
240 023F E8         C2     INX
250 0240 BD8302            LDA  TBL,X    ;GET NEXT TBL CHAR.
260 0243 D0FA              BNE  C2       ;LOOP IF <> 0
270 0245 BD8402            LDA  TBL+1,X  ;DOUBLE NULL?
280 0248 D0E8              BNE  C0       ;NO, NEXT TBL ENTRY
290 024A            ;
300 024A A018       PKLIN  LDY #LEN      ;LINE LENGTH OF SCREEN
310 024C B145              LDA  (ST),Y   ;GET CHAR TO BE MOVED
320 024E AA                TAX           ;SAVE CHARACTER
330 024F A920              LDA #$20      ;ERASE OLD CHARACTER
340 0251 9145              STA  (ST),Y
350 0253 8A                TXA           ;RESTORE CHARACTER
360 0254 9147              STA  (PP),Y   ;PRINT AT NEW LOCATION
370 0256 88                DEY
380 0257 10F3              BPL  PKLIN+2
390 0259            ;
400 0259 38         DECPP  SEC           ;POKE POINT UP 1 LINE
410 025A A547              LDA  PP
420 025C E920              SBC #LINE
430 025E 8547              STA  PP
440 0260 B002              BCS  DECST
450 0262 C648              DEC  PP+1
460 0264            ;
470 0264 38         DECST  SEC           ;START UP 1 LINE
480 0265 A545              LDA  ST
490 0267 E920              SBC #LINE
500 0269 8545              STA  ST
520 026B C646              DEC  ST+1
530 026D            ;
540 026D A546       DONE   LDA  ST+1     ;ST <$D000 IF DONE
550 026F C9D0              CMP #$D0
560 0271 B0BD              BCS  CKLIN
570 0273 A962              LDA #$62      ;POINT MESS. AT $A162
580 0275 4C78A2            JMP  $A278    ;TO WARM START
590 0278            ;
600 0278 A018       ERASE  LDY #LEN      ;ERASE A LINE
610 027A A920              LDA #$20
620 027C 9145              STA  (ST),Y
630 027E 88                DEY
640 027F 10FB              BPL  ERASE+4  ;LOOP IF NOT DONE
650 0281 30E1              BMI  DECST    ;GO TO DECST IF DONE
660 0283            ;
670 0283            TBL
680 0283 20         .BYTE $20,$20,$20,$20,0
680 0284 20
```

```
.BYTE '?',0,'OK',0,'LIST',0,0

680 0285 20
680 0286 20
680 0287 00
690 0288 3F
690 0289 00
690 028A 4F
690 028B 4B
690 028C 00
690 028D 4C
690 028E 49
690 028F 53
690 0290 54
690 0291 00
690 0292 00
```

THOMAS W. KELLAR, OHIO

This program is written for those rare systems with the English ROM and 23K or more memory and an MX-80 printer. It is a screen editor that puts about three pages-single spaced-into memory and allows you to edit it. It also does underlining, but that is just an added gimic. The program follows none of your rules for REMs, indeed it has none of those things. However, it is free, and my rules suit me.

The conventions for it are:

All keys initiated by holding down CTRL simultaneously.

i wait until second keystroke and put that up on the screen-used to implement control keys as data
o output the RAM buffers to the printer
z delete character on screen and fill right hand side with non printing nulls
c don't use this as this is the standard CTRL/C to stop the program
n display the previous page of memory and update the current one
p display the previous page
d move cursor nondestructively down the screen
u same as d but up
l same as u but left
r same as l but right-l and r wrap the cursor around the screen
(RUBOUT is the same as the CTRL/l sequence)
x non working function that is supposed to insert a line of text
b starting and ending marker for underlining of text
v display the non printing nulls on the screen-as opposed to printable blanks
(CTRL/v will annihilate the underline characters.)

I thought about implementing the real time clock to give the thing a blinking cursor, but then decided nah. The non printing nulls are binary two's and are put on the screen as 96 as opposed to blanks which are 32. The program stores about 13 screens worth of data in memor and could easily be modified to use all of 32K memory. A RETURN is necessary to terminate print lines, but typing automatically wraps around the screen so there is not really any "word wrap". So about every three screen lines I hit the RETURN-the print logic generates the Line Feeds so only the RETURN is necessary. The underlines are generated by counting characters and doing a RETURN then printing blanks then printing underscores to generate the underlines. Oh, yes the CTRL/V is turned off by typing a second character.

```
1 A=0:R=533:M=0:B=6000:Q=0:I=0:J=0:M=0:F=576:N=0:O=1:PRINTCHR$(26);
2 FORI=BTOB+8200:POKEI,96:PRINTI;CHR$(13);:NEXTI:V=24:S=53412:T=32
3 U=23:E=0:GOTO10:REM TWK EDITOR V5.8 7 September 1981
7 FORJ=ETOU:FORI=ETOU:A=S+I+T*J:POKEA,96:NEXTI:NEXTJ:RETURN
10 PRINT"0          Page";CHR$(13);:GOSUB7:POKE11,0:POKE12,253
11 A=S:Z=A
12 I=PEEK(A):POKEA,60:M=USR(M):M=PEEK(R):IFM=26ORM=9ORM=VTHEN600
13 IFM=4ORM=21ORM=18ORM=12ORM=95THEN20
14 IFM=14ORM=15ORM=16THEN40
15 IFM=22THENPOKEA,I:GOSUB800:GOTO12
16 POKEA,M:IFM=13THENZ=Z+T:A=Z:GOTO12
17 A=A+O:IFA>Z+UTHENZ=Z+T:A=Z:GOTO12
18 GOTO12
20 IFM<>4THEN23
21 POKEA,I:A=A+T:Z=Z+T:IFA>54179THENA=A-T:Z=Z-T:GOTO12
22 GOTO12
23 IFM<>21THEN26
24 POKEA,I:A=A-T:Z=Z-T:IFA<STHENA=A+T:Z=Z+T:GOTO12
25 GOTO12
26 IFM<>18THEN29
27 POKEA,I:A=A+O:IFA>Z+23THENA=Z:GOTO12
28 GOTO12
29 POKEA,I:A=A-O:IFA<ZTHENA=Z+U:GOTO12
30 GOTO12
40 IFM<>14THEN45
41 POKEA,I:Z=T:Q=Q+O:IFQ>13THENQ=Q-O:GOTO11
42 FORJ=ETOU:FORI=ETOU:A=S+I+Z*J:N=PEEK(A):M=PEEK(B+Q*F+I+J*V)
43 POKEB+(Q-O)*F+I+J*V,N:POKEA,M:NEXTI:NEXTJ:A=S:Z=A
44 PRINTQ;CHR$(13);:GOTO12
45 IFM<>16THEN500
46 Z=T:POKEA,I:Q=Q-O:IFQ<ETHENQ=Q+O:GOTO11
47 FORJ=ETOU:FORI=ETOU:A=S+I+Z*J:N=PEEK(A):M=PEEK(B+Q*F+I+V*J)
48 POKEA,M:POKEB+(Q+O)*F+I+J*V,N:NEXTI:NEXTJ:A=S:Z=A
49 PRINTQ;CHR$(13);:GOTO12
500 G=0:H=0:POKEA,I:M=0:PRINTCHR$(26)
501 DB=0:INPUT"Line spacing";I:IFI>2ORI<1THEN501
510 PRINT"ESC TO START":IFI=2THENDB=-1
530 IFPEEK(57088)<>222THEN530
540 SAVE:PRINT:PRINT:PRINT:PRINT:PRINT;
550 NL=0:FORI=0TO13:FORJ=0TO23:FORK=0TO23
579 L=PEEK(B+I*F+K+J*V):IFL=5THENPOKE517,0:PRINTCHR$(26);:GOTO10
```

```
580 IFL<>96THENGOSUB910:C=C+1
581 IFL=13THENPRINTCHR$(10);:NL=NL+1:C=0:IFNL=56THENGOSUB700
583 IFDB=-1THENIFL=13THENPRINTCHR$(10);:NL=NL+1:IFNL>56THENGOSUB700
584 IFL=2ANDG<>1THENG=1:GOTO590
585 IFG=1ANDL<>2ANDL<>96THENH=H+1:GOTO590
586 IFG=1ANDL=2THENGOSUB900:G=0:H=0
590 NEXTK:NEXTJ:NEXTI:PRINTCHR$(140):POKE517,0:PRINTCHR$(26);:GOTO10
600 IFM<>26THEN650
605 FORN=ATOZ+U:J=PEEK(N+O):POKEN,J:NEXT:POKEZ+U,96:GOTO12
650 IFM<>9THEN750
651 M=USR(M):M=PEEK(R):GOTO16
700 NL=0:PRINTCHR$(140);:PRINT:PRINT:PRINT:PRINT:PRINT;:RETURN
750 POKEA,I:FORJ=B+14*576-UTO(Q+O)*F+BSTEP-V:FORI=ETOV:P=PEEK(J+I)
752 POKEJ+I+V,P:NEXTI:PRINTJ;CHR$(13);:NEXTJ
753 FORJ=54148TO54148+U:P=PEEK(J):POKE(Q+O)*F+B,P:NEXTJ
756 PRINTQ"   ";CHR$(13);:GOTO12
800 FORJ=ETOU:FORI=ETOU:M=S+I+T*J:IFPEEK(M)=96THENPOKEM,3
802 NEXTI:NEXTJ:I=USR(I)
804 FORJ=ETOU:FORI=ETOU:M=S+I+T*J:IFPEEK(M)=3THENPOKEM,96
806 NEXTI:NEXTJ:RETURN
900 C=C-2:PRINTCHR$(13);:FORA=1TOC-H:PRINTCHR$(32);:NEXTA
902 FORA=1TOH:PRINTCHR$(95);:NEXTA:RETURN
910 A=L:M=PEEK(61440):IF(MAND1)=0THEN910
915 POKE61441,A:RETURN
OK
```

CORRECTION TO ALIEN RAIN

by Nelson Van Elderen, Michigan

The following is a list of lines to add or change in Billy D. Smith's ALIEN RAIN program to add sound to it. (Aardvark Journal Vol. 2, No. 5, Page 15.) I encountered one problem while working on this. Every time I tried to Add or Change a line, line 170 would scramble itself. Separating line 170 into two separate lines seemed to fix the problem. I say seemed to because I have absolutely no idea why the problem would occur in the first place, nor why splitting the line into two lines would fix it. I also wonder if this would be a problem unique to my system (C2/4P w/8K), a bad memory chip somewhere, or if it is a problem other have run accross.

Also, I don't know if this the best or most efficient way of adding sound to a program, but with limited resources I had (OSI's Manuals) it was the best I could come up with.

TO ADD ARCADE TYPE SOUND TO "ALIEN RAIN"
C4P

ADD - CHANGE FOLLOWING LINES:

```
6 Q=57089:POKEQ,1:POKE56832,2:I=1
45 POKEMP,U:MP=MP-V:IFPEEK(MP)=UTHEN
POKEMP,F:POKEQ,150:POKEQ,I:GOTO45
56 FORX=100TO150:POKEQ,X:POKEQ,I
95 IFPM=TGTHENSC=SC+40:DB(CY)=-1:
FORX=75TO150:POKEQ,X:NEXT:POKEQ,I
96 GOTO120
136 FORX=0TO50:POKEQ,X:POKEQ,X+50:
NEXT:POKEQ,I
172  FORX=1TO8:FORY=150TO200STEP5:
POKEQ,Y:NEXTY:POKEQ,I:NEXTX:POKEQ,I
191 FORX=150TO200:POKEQ,X:NEXT:POKEQ,I
206 FORX=1TO8:FORY=150TO200STEP5:
POKEQ,Y:NEXTY:POKEQ,I:NEXTX:POKEQ,I
342 IFPP<128THENV=64:BA=53573:L=5:R=3
351  FORX=1TO3:FORY=150TO210:POKEQ,Y:
POKEQ,Y-20:NEXTY:NEXTX
352 POKEQ,I
445 FORX=50TO200:POKEQ,X:NEXT
446  FORX=1TO20:FORY=10TO200STEP10:
POKEQ,Y:NEXTY:NEXTX:POKEQ,I
447  PRINT:PRINT:PRINT:PRINT"TOTAL
SCORE=";SC
```

ROBERT VAN SINGEL, 3 RIVERS, MICHIGAN

```
10   INPUT"WHAT DO YOU WANT THE BUNNY TO
SAY";A$
20  A=LEN(A$):IFA>10THENDIMB(A-1)
30   FORI=0TOA-1:B$=MID$(A$,1+I,1):
B(I)=ASC(B$):NEXT
40   INPUT"OUTPUT ON PRINTER OR TERMINAL
(P/T)";A$
50  IFA$<>"P"THENZ=10:GOTO70
60  DISK!"IO ,01":Z=15
70  GOSUB140
80  PRINT
90  READX:IFX<0THEN80
100 IFX>128THEN130
110 PRINTTAB(X+Z);:READY:FORI=XTOY:
J=I-A*INT(I/A)
120 PRINTCHR$(B(J));:NEXT:GOTO90
130 GOSUB140:GOTO290
140 FORI=1TO6:PRINTCHR$(10);:NEXT:
RETURN
150  DATA1,2,-1,0,2,45,50,-1,0,5,43,
52,-1,0,7,41,52,-1
160 DATA1,9,37,50,-1,2,11,36,50,-1,
3,13,34,49,-1,4,14,32,48,-1
170  DATA5,15,31,47,-1,6,16,30,45,-1,
7,17,29,44,-1,8,19,28,43,-1
180 DATA9,20,27,41,-1,10,21,26,40,-1,
11,22,25,38,-1,12,22,24,36,-1
190  DATA13,34,-1,14,33,-1,15,31,-1,17,
29,-1,18,27,-1
200  DATA19,26,-1,16,28,-1,13,30,-1,11,
31,-1,10,32,-1
210  DATA8,33,-1,7,34,-1,6,13,16,34,-1,
5,12,16,35,-1
220 DATA4,12,16,35,-1,3,12,15,35,-1,2,
35,-1,1,35,-1
230 DATA2,34,-1,3,34,-1,4,33,-1,6,33,
-1,10,32,34,34,-1
240 DATA14,17,19,25,28,31,35,35,-1,15,
19,23,30,36,36,-1
250  DATA14,18,21,21,24,30,37,37,-1,13,
18,23,29,33,38,-1
260 DATA12,29,31,33,-1,11,13,17,17,19,
19,22,22,24,31,-1
270 DATA10,11,17,18,22,22,24,24,29,29,
-1
280 DATA22,23,26,29,-1,27,29,-1,28,29,
-1,4096
290 IFA$="T"THENEND
300  PRINTCHR$(27):CHR$(70):  DISK!"IO
,02":END
```

CORRECTION FOR YACHT RACE
by John Sakamoto, California

The following corrections were made to the "Yacht Race" program that was published in the February 1982 (Vol. 2, No. 6) issue of the Aardvark Journal.

```
101 POKEMA-1,65:POKEMB+32,66:
POKEMC+1,67:POKEMD-32,68
537 IFAM$="N"THENAM=-MN(3)
607 IFBM$="N"THENBM=-MN(3)
704 IFCM$="N"THENCM=-MN(3)
```

WARD HORNER, MARYLAND

I own a C1P modified to run at 2 Mhz and have a passion for programming realtime games. I program both in BASIC and machine code and enjoy designing interesting graphics routines.

I am amazed at the number of unrealistic explosion routines I have seen. Most of these routines bear only a vague resemblance to an explosion and they don't even begin to explore the vast possibilities of OSI graphics.

With this in mind, I have written an explosion routine from my latest game (which is called HITMAN). I modified it to run on C2/4 machines and added a control loop. It was designed to run on a 2 Mhz machine, but it still looks pretty good at 1 Mhz. To start explosion, press shift.

The graphics in this routine may be easily changed by modifying line 1000 and changing SE (size of explosion) in line 20, to the number of characters in your modified explosion. Also, try incrementing TE (type of explosion) in line 20 for some interesting effects.

In HITMAN, I set TE=2. This setting leaves a few dots behind, as you will see, but if you use dots for background stars or make the last two characters in the explosion blanks it won't matter.

```
10  FORI=1TO30:PRINT:NEXT
20    SE=14:TE=0:T2=32:DIME(SE):
FORI=1TOSE:READE(I):NEXT
40  CT=53775:A=31:B=32:C=33:NL=254
60  IFPEEK(57088)>128THEN100
80  CT=54238:A=63:B=64:C=65:NL=1
100  RW=INT(9*RND(8)-4):CL=INT(9*RND(8)
-4):M=CT+RW+CL*B:POKEM,241
120 IFPEEK(57100)=NLTHEN120
199 REM explosion routine
200 FORI=1TO5:FORJ=16TO32:POKEM,J:
FORK=1TO3:NEXTK,J,I
220 FORI=1TOSE
240 POKEM+A*I,E(I):POKEM-A*I,E(I):
POKEM+C*I,E(I):POKEM-C*I,E(I)
260 POKEM-I,E(I):POKEM+I,E(I):
POKEM+B*I,E(I):POKEM-B*I,E(I):J=I-TE
280  POKEM+A*J,T2:POKEM-A*J,T2:POKE
M+C*J,T2:POKEM-C*J,T2
300  POKEM-J,T2:POKEM+J,T2:POKE
M+B*J,T2:POKEM-B*J,T2:NEXT
500 GOTO100
1000 DATA182,13,214,4,60,43,199,
198,127,47,44,44,46,46
```

NOTE: If you set TE=Z and don't want to leave dots behind, add to the end of line 1000 ",32,32" and make SE in line 20 equal to 16.

A LITTLE NOTE FROM CHRIS DAVIES

You would think that living in New Zealand would put us right out of the computer scene, this is NOT true. Although a Superboard here costs over NZ $600 there are quite a few people with OSI machines and we even have our own OSI users group here in Christchurch. (Note: how do we get official recognition?)

I have an 8K Superboard II and am very pleased with it. Incidentally my board was bought in the States for U.S. $279.00. (A considerable saving). Upgrading to a C4PMF is a distinc possiblity because the 8K of the Superboard just is not sufficient! Well that's about all I have to say.

I just thought I'd right and let you know that even we Kiwis have computers and also that New Zealand is not an Aussie state (as some people seem to think. Have a good look at a world map sometime you might even find us!) Having encountered tips in your journal and from our group has made programming a relative breeze after working on a Digital machine for four years. I follow your Journal with much interest.

CORRECTION by Alan Falkenstein

I have finally solved the Death Ship adventure game. (After about 30 hours of work). I enjoyed it throughly. It certainly was a well written game. I find it truly amazing that you were able to do it within 8K of memory. I have made a few changes that you may wish to use.

1) STRING BUG FIX: change DIM (line 110) of D$ from 18 to 20. In line 120, insert Q9=FRE(X) in front of the prints. (reference: page 28 of Ed Carlson's "All About OSI Microsoft Basic In ROM)

2) DOOR IN CAPTAINS CABIN: After entering the Cabin, there are no exits and no door. I chose to change the 2nd 12 in line 740 to a 14.

3) ROPE TIED TO RAIL?: After you get the water in the bucket, you'll observe that the rope is tied to "BU" if you go to the MIDSHIP DECK. I can't explain why, but I changed lines 1400-1430:

```
1400 PRINT"TO WHAT":GOSUB1690:GOTO1420
1410 O$(8)="ROPE TIED TO RAIL":L(2)=L:
GOTO120
1420 IF(A$="ST"ORA$="RA")ANDL=6ANDL(2)=0
THENF2=1:GOTO1410
1430 PRINT"NOT THERE, STUPID":GOTO120
```

A FEW CHANGES IN 'BREAKTHRU'
by Jay Friedman, Ohio

The following program changes in Aardvark's BREAKTHRU allow for additional walls to be played after the first wall has been broken down. Since it is fairly difficult to eliminate each and every block for a score of 420, line 340 can be modified to build a new wall at a smaller score. For example, IF CH<400 then 350. This way four of five blocks can be left then a new wall is poked onto the screen.

```
165 FORX=SD-3TOSD+3:POKEX,32:NEXT
166 REM CLEARS OLD SCORE
195
FORX=GC+OFTOGC+OF+2040:POKEX,13:NEXT:
GOSUB200:GOTO230
196 REM  200  IS THEN  WALL BUILDING
ROUTINE
220 POKEGC+X+5*LC,161:POKEGC+X+6*LC,187:
NEXT:CH=0:RETURN
221 REM  CH  KEEPS  SCORE  OF CURRENT
SCREEN
290 D$=STR$(S):D=SD-3
300 FORY=2TOLEN(D$):POKED+Y,ASC(MID$
(D$,Y,1)):NEXT
310 POKESD+12,BALLS+48
311 REM  290-300 POKES  NEW SCORE ONTO
SCREEN. 310 POKES BALL NUMBER
340 P=PEEK(57088):IFCH<>420THEN350:
345  IFB<GC+3*LC+2ORB>GC+7*LC+W-1THEN
GOSUB200
346 REM 420 IS THE SCORE FOR ONE SCREEN
347  REM  345 WAITS UNTIL BALL IS OUT OF
WALL AREA BEFORE GOING TO 200
870 IFP=187THENS=S+2:CH=CH+2
880 IFP=161THENS=S+1:CH=CH+1
890 IFP=232THENS=S+4:CH=CH+4
900 IFP=233THENS=S+6:CH=CH+6
901 REM 870-900 ADD POINTS TO CH
```

RUSS TERRELL, ROSEMOUNT, MINNESOTA
CORRECTIONS

These program patches will let you use Joysticks with the game "ANTI-AIRCRAFT ARTILLERY" Vol. 2, No. 2.

```
120  CO=53284:LC=32:W=25:P1=191:P2=127:
FORX=0TO6:READK(X):NEXT:TA=1
900 DATA4,8,2,64,128,32,2,127,247,
223,127,247,223,253
```

After 10 hours of work, here are the program patches to get monitor ROM. Due to conflicting addresses in ROM and the program, BASIC crashed. The program has been relocated to $0232 (562).

```
5 POKE621,0:FORK=627TO641:POKEK,32:
NEXTK:POKEK,255
10 DATA562,620
20 DATA174,59,211,24,173,51,2,105,32,
141,71,2,173,52,2,105,0
30  DATA141,72,2,142,195,208,24,173,51,
2,201,195,208,8,173,52,2
40 DATA201,208,208,1,96,56,173,51,2,
233,1,141,51,2,173,52,2
50 DATA233,0,141,52,2,76,50,2
60 **this line has been deleted.
70 PRINTCHR$(26);"  Potato Chips"
120 POKE11,0:POKE12,253:X=USR(X):K=627
124  X=PEEK(K):POKE53413+K-627,X:K=K+1:
GOTO123
190  POKEP,127:FORZ=1TO10:IFPEEK(P)<>
127THEN250
1095  PRINT:PRINT:PRINT"The  high  score
is"; PEEK(621):PRINT
1097 IFS>PEEK(621)THEN1150
1150 PRINT"You  have beaten
it.":PRINT:PRINT:POKE621,S:INPUT"Your
name"
1170  FORK=627TO627+LEN(S$)-1:POKEK,ASC
(MID$(S$,K-626,1)):NEXTK
1530 DATA141,114,2,169,2,133,12,169,50,
133,11,169,59,141,51,2
1540 DATA169,211,141,52,2,96
```

JACK VAUGHN, TEXAS

Here is a short program called "TEST YOUR REFLEXES". I have no way to test it except on my Superboard, but I believe it will work on others.

```
 2 REM JACK VAUGHN
 4 REM 3695 BRYAN DRIVE
 6 REM BEAUMONT, TEXAS 77707
 8 REM LINES 10-18 SET UP FAST SCREEN CL
EAR CALLED BY 'X=USR(X)'
 10 FORX=576TO598:READY:POKEX,Y:NEXT
 12 DATA169,208,160,0,133,255,132,254
 14 DATA162,4,169,32,145,254,200,208,251
 16 DATA230,255,202,208,246,96
 18 POKE11,64:POKE12,2
 22 C$="SWHSOIUN PDGEERRRAMWNAODNMMFAAAN
SSOPTUESDEYKDORY"
 23 VIDEO=600:IFPEEK(57088)<129THENVIDEO
=540:GOTO26
 24 P1=247:KP=189:PO=530:GOTO28:KP=VALUE
 IF D&K ARE PRESSED TOGETHER.
 26 P1=8:KP=66:PO=2073
 28 X=USR(X):REM SCREEN CLEAR
 29 PRINT"  TEST YOUR REFLEXES":PRINT:PR
INT:PRINT:PRINTSPC(10)"OR
 30 PRINT:PRINT:PRINT:PRINTSPC(4)"ARE YO
U SOBER?"
 31 PRINT:PRINT:PRINT:FORX=1TO3000:NEXT
 36 PRINT:PRINT"  AS SOON AS YOU SEE":PR
INT:PRINT"        ** GO **,"
 38 PRINT:PRINT"  PRESS  D   AND   K
 40 PRINT:PRINT"  AT  THE SAME TIME.":PR
INT
 42 INPUT"  READY";A$
 46 X=USR(X)
 48 FORX=1TO4500*RND(X):NEXT
 50 FORX=1TO12:PRINT:NEXT:PRINTSPC(8)"**
 GO **
 52 FORX=0TO10000:POKEPO,1:POKE57088,P1:
PE=PEEK(57088)
 54 IFPE=KPGOTO58
 56 NEXT
 58 POKEPO,0:PRINT:PRINTSPC(10)X
 60 PRINT:PRINT:IFX>3GOTO66
 62 PRINTSPC(5)"YOU CHEATED":PRINT:PRINT
SPC(5)"DO IT AGAIN
 64 GOTO48
 66 IFX<8GOTO100
 68 IFX<11GOTO110
 70 IFX<13GOTO120
 72 IFX<15GOTO140
 74 IFX<17GOTO130
 76 PRINT:PRINT
 78 PRINT" SOBER UP AND TRY AGAIN!
 80 PRINT:PRINT:PRINT:PRINT
 82 PRINT" TRY AGAIN";:INPUTA$
 84 X=USR(X):FORX=1TO6:PRINT:NEXT:GOTO46
 86 PRINTSPC(3)"TEST YOUR REFLEXES
 100 PRINTSPC(8);:FORY=4TO25STEP3:PRINTM
ID$(C$,Y,1);:NEXT
 102 GOTO80
 110 PRINTSPC(7);:FORY=2TO32STEP3:PRINTM
ID$(C$,Y,1);:NEXT
 112 GOTO80
 120 PRINTSPC(10);:FORY=28TO37STEP3:PRIN
TMID$(C$,Y,1);:NEXT
 122 GOTO80
 130 PRINTSPC(7);:FORY=3TO30STEP3:PRINTM
ID$(C$,Y,1);:NEXT
 132 GOTO80
 140 PRINTSPC(9);:FORY=33TO48STEP3:PRINT
MID$(C$,Y,1);:NEXT
 142 GOTO80
 200 REM JACK VAUGHN
 202 REM 3695 BRYAN DRIVE
 204 REM BEAUMONT, TEXAS 77707
```

** PERSONAL ADS **

FOR SALE: OSI C1P, 32K DISK DRIVE, C1E ROM, PROG. SOUND GENERATOR, JOYSTICKS, VIDEO MONITOR, VIDEO MODD II FOR 32 CHAR. DISPLAY, RS232, OS65D V3.3 AND HEXDOS OP SYSTEMS, COMPLETE DOCUMENTATION, COST $1300+. BEST OFFER OVER $800. R. WHITAKER, 3715 RIDGECREST DR., SALT LAKE CITY, UT 84118

FOR SALE: C2/8P 8K, 540 VIDEO, 502 CPU, 580-8 SLOT BACKPLANE, AARDVARK JOYSTICKS, 12" B/W MONITOR, SOFTWARE AND DOCUMENTATION. $650. CALL (616) 399-3109 CRAIG

FOR SALE: 48K C4P-MF, 2 DISK DRIVES, D&N EXPANSION INTERFACE, 8 CARD BACKPLANE, 24K MEMORY BD (ALL NEC CHIPS), MICROTEK SERIAL IMPACT PRINTER. SOFTWARE INCLUDES OSI PLANNER+, OSI MDMS, DWO QUONG WP6502, OS WP2, AARDVARK SUPERDISK, OS65D V3.3, SAMS MANUAL. $2400. D. BROUDY (602) 729-5459 EVENINGS.

FOR SALE: OSI C2/4P BASIC-IN-ROM (8K RAM) 2 YRS OLD, USED ONLY SEVERAL MTHS. COMPLETE W/ALL MANUALS AND DOC., CABLES, DEMO-PROGRAM TAPE. SHIPPED VIA UPS. $560. CONTACT: AL ADAMS, 4512 N. SAGINAW RD, APT #221C, MIDLAND,MI 48640

FOR SALE: OSI C3C 36 MB. BEST OFFER. CALL (312)835-4456

FOR SALE: OSI C3B 74M H.D. 3 MICRO-TERM CRT. T.I. 820 PRINTER SPEED .7 MIPS 1 YR OLD. 16,500. (313) 342-1020 / BOX 517, ROYAL OAK, MI 48068

FOR SALE: 5 YR. COLLECTION OF OSI EQUIP. INCLUD. (2) C1P'S, A BROKEN 610 BD AND (2) DISK DRIVES. THERE ARE MANY MORE ITEMS AND MANY ORIGINAL PROGRAMS BOTH ON DISK AND TAPE. PLEASE SEND SASE FOR A LISTING AND PRICE TO E.H. BROWN, P.O. BOX 2211, WARNER ROBINS, GA 31099.

FOR SALE: SUPERBOARD PRE-CUT CABINET KIT $27.95 PPD. COMP. W/HARDWARE & EXCEL INST SET, RS232 KIT $9.95 PPD. WRITE FOR CAT OF KITS, HARDWARE AND ACCES TO DEE PRODUCTS, 150A BIRCHWOOD, LK MARION, ILL 60110.

FOR SALE: OSI SUPERBOARD 2 18K. GRAFIX SUPER EXPANSION BD, HI-RES DISPLAY, PARALLEL PORT & EXPANSION RAM. CASE & MUCH SOFTWARE. ASKING $450. STUART HAAS, (914) 357-3447

WANTED: AMATEUR ASTRONOMER LOOKING FOR ASTRONOMY PROGRAMS THAT RUN ON A C1P. IF ANYONE IS USING THE C1P FOR TELESCOPE CONTROL, ANY IDEAS WOULD BE HELPFUL. ANYONE THAT CAN HELP WRITE: DON GRENDA, 3333 5TH AVE, UNIT 6D, S. MILWUAKEE, WI 53172

FOR SALE: OSI C4P-MF, 48K, 4 DUAL-SIDED MF DRIVES, 64 X 32, 9" B/W MONITOR, HI-CAPACITY SWITCHING POWER SUPPLY, ON SCREEN CLOCK -CALENDAR, W/BATTERY, BSR INTERFACE, (2) JOYSTICKS, VOTRAX VOICE SYN., TTY MODEL 40 LINE PRINTER, OTHER ACCES., COMP. DOC., LOTS OF SOFTWARE, $3000. BOB ARONSON, (516) 799-3679

AARDVARK
2352 SOUTH COMMERCE
WALLED LAKE,MI 48088

BULK RATE
U. S. POSTAGE
PAID
WALLED LK.,MI.
Permit No. 5